RHCSA Crash Course

RHEL 9

Sander van Vugt

Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

Audience Poll Question

Rate your Linux experience? (Single response)

- 0
- 1
- 2
- 3
- 4
- 5

Audience Poll Question

Where are you from?

- Africa
- Middle East
- India
- Asia (others)
- Australia/Pacific
- North/Central America
- South America
- Europe

Day 1 Agenda

1. Software Management
2. Getting Started
3. su and sudo
4. Partitions

Day 2 Agenda

1. Users and Groups
2. Permissions
3. Networking
4. Processes
5. Systemd
6. Boot Procedure

Day 3 Agenda

1. LVM
2. Stratis
3. Remote Mounts and Autofs
4. SELinux
5. Firewalling
6. Time
7. Scheduling Tasks

Day 4 Agenda

1. Troubleshooting
2. Containers
3. Optional Exam Practice Lab

Additional Resources

- RHCSA RHEL 9 Complete Video Course
- RHCSA 9 Cert Guide
- See resources.txt in the course Git repository at https://github.com/sandervanvugt/rhcsa9 for more resources

Required Setup

- **Recommended:** install a virtual machine
  - Any installation of RHEL 9 or later
  - Option to add disk space later
  - Warning: do NOT install any software on your newly installed virtual machine!
- **Not recommended**: because of missing functionality
  - Cloud VM
  - O'Reilly Sandbox

Managing Software

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

Understanding RPM Packages

- Software on RHEL is installed using packages in Red Hat Package Manager (RPM) format
- To handle dependency management, RHEL uses repositories for package installation
- To install software from repositories, **dnf** is used
- Installed packages are registered in the RPM database

Understanding Repositories

- A repository is a collection of RPM package files with an index that contains the repository contents
- Repositories are often offered through web sites, but local repositories can be created also
- The **dnf** command is used as the default command to install packages from repositories
- In RHEL 9, **dnf** is preferred over the **yum** command which was used in previous versions of RHEL
- **dnf** and **yum** are offering the same functionality

Demo: Configuring the Installation Disk as Repo

This procedure is NOT required on the exam. You'll need it to set up your own lab environment

- **df -h** # You need 10 GB free disk space on /
- **dd if=/dev/sr0 of=/rhel9.iso bs=1M**
- **mkdir /repo**
- **cp /etc/fstab /etc/fstab.bak**
- **echo "/rhel9.iso /repo iso9660 defaults 0 0" >> /etc/fstab**
- **mount -a**

If you don't have 10 GB free space in /

- **mkdir /repo**
- **echo "/dev/sr0 /repo iso9660 defaults 0 0" >> /etc/fstab**

Accessing Repositories

- To access repositories, a RHEL system must be registered using **subscription-manager**
- **subscription-manager** tries to access the online Red Hat repositories
- As an alternative to online repositories, repositories can be offered through Red Hat Satellite
- If no Internet connection, nor Red Hat Satellite are available, no repositories will be available by default
- In that case, you'll have to manually configure repository access

Manually Configuring Repository Access

- Repositories access is configured through repo files in /etc/yum.repos.d/, or using **dnf config-manager**
  - **dnf config-manager --add-repo="file:///repo/AppStream"**
  - **cat >> /etc/yum.repos.d/AppStream.repo <<EOF**
    > [AppStream]
    > name=AppStream
    > baseurl=file:///repo/AppStream
    > gpgcheck=0
    EOF
- Make sure you know this on the exam!

Using GPG Keys

- To ensure that packages have not been tampered with, GPG keys can be used
- A repository GPG key is used to sign all packages and before installing the package, its signature is checked
- To do this, you'll need a local GPG key to be present
- To make accessing trusted repositories easier, use the **gpgcheck=0** option in the repository client file

Managing Packages with **dnf**

dnf was created to be intuitive

- **dnf list** lists installed and available packages
  - **dnf list 'selinux*'**
- **dnf search** searches in name and summary. Use **dnf search all** to search in description as well
  - **dnf search seinfo**
  - **dnf search all seinfo**
- **dnf provides** searches in package file lists for the package that provides a specific file
  - **dnf provides */Containerfile**
- **dnf info** shows information about the package

Managing Packages with **dnf**

- **dnf install** installs packages as well as any dependencies
- **dnf remove** removes packages, as well as packages depending on this package - potentially dangerous!
- **dnf update** compares current package version with the package version listed in the repository and updates if necessary
  - **dnf update kernel** will install the new kernel and keeps the old kernel as a backup

Understanding Groups

- A dnf group is a collection of packages
- A regular group is just a collection of packages
- An environment group is used to install a specific usage pattern, and may consist of packages and groups
- Use **dnf group list** to see a list of groups
- Some groups are normally only installed through environment groups and not separately, and for that reason don't show while using **dnf group list**
- Use **dnf group list hidden** to see these groups as well

Lab: Managing Software

- Ensure your system is using a repository for base packages as well as application streams
- Find the package that contains the seinfo program file and install it
- Download the httpd package from the repositories without installing it, and query to see if there are any scripts in it

Getting Started

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

Lab 1: Getting Started Lab

- Use tar to create a gzipped archive that contains the contents of the /etc directory as relative file names, and write the archive to /root/archive.tgz
- Filter all usernames (in the first column) from /etc/passwd in alphabetical order and with no blank lines in the target file and write them to /var/tmp/users
- Create a user with the name bob. Next, find all files that are owned by user bob and copy them to /root/userfiles

Use **live-lab1-grade.sh** from the labs directory in the course Git repository at https://github.com/sandervanvugt/rhcsa9 to grade your work

su and sudo

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Using **su -**

- The **su** command is used to switch current user account from a shell environment
- If **su** is used with the **-** option, the complete environment of the target user is loaded
- While using **su**, the password of the target user is entered
- If the root user has a password, **su -** can be used
- Using **su -** to open a root shell is considered bad practice, use **sudo -i** instead
- The **su** command can be useful for testing other user accounts

Using **sudo**

- **sudo** is a more secure mechanism to perform administration tasks
- Behind **sudo** is the /etc/sudoers configuration file
- While editing /etc/sudoers through **visudo**, very detailed administration privileges can be assigned
- To run an administration task using **sudo**, use **sudo command**
- This will prompt for the current user password, and run the command if this is allowed through /etc/sudoers
- To open a root shell, **sudo -i** can be used

Providing Administrator Access

- Users that are a member of the group **wheel** get full sudo access
  - This is accomplished by **%wheel ALL=(ALL) ALL** in /etc/sudoers
  - Use **usermod -aG wheel myuser** to add a user to the group wheel
- Do NOT enable the line **%wheel ALL=(ALL) NOPASSWD: ALL**
  - It will provide full sudo access without entering a password and is very dangerous
- If you don't like entering your user password every five minutes, increase authentication token expiration by adding the following
  - **Defaults timestamp_type=global,timestamp_timeout=60**

Providing Access to Specific Tasks

- Use drop-in files to provide admin access to specific tasks
  - **lisa ALL=/sbin/useradd, /usr/bin/passwd**
- Consider using command arguments to make the commands more specific
  - **%users ALL=/bin/mount /dev/sdb, /bin/umount /dev/sdb**
  - **linda ALL=/usr/bin/passwd, ! /usr/bin/passwd root**

Managing Partitions

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Linux Storage Options

- Partitions
  - Used to allocate dedicated storage to specific types of data
- LVM Logical Volumes
  - Used at default installation of RHEL
  - Adds flexibility to storage (resize, snapshots and more)
- Stratis (no longer in the exam objectives)
  - Next generation Volume Managing Filesystem that uses thin provisioning by default
  - Implemented in user space, which makes API access possible

Linux Block Devices

- According to the driver that is used, different block devices can be used
- Use **lsblk** to get an overview of currently existing devices
- **/dev/sdx** is common for SCSI and SATA disks
- **/dev/vdx** is used in KVM virtual machines
- **/dev/nvmexny** is used for NVME devices

Understanding Partition Numbering

- On **sd** and **vd** devices, partitions are numbered in order:
  - /dev/sda2 is the second partition on the first hard disk
  - /dev/sdb1 is the first partition on the second hard disk
- On **nvme** devices, partition names are appended to the device name
  - /dev/nvme0n1p1 is the first partition on the first hard disk
  - /dev/nvme0n3p2 is the second partition on the third hard disk
- Use **lsblk** to avoid confusion!
  - **lsblk** reads the kernel partition table in /proc/partitions

Understanding GPT and MBR Partitions

- Master Boot Record (MBR) is part of the 1981 PC specification
  - 512 bytes to store boot information
  - 64 bytes to store partitions
  - Place for 4 partitions only with a max size of 2 TiB
  - To use more partitions, extended and logical partitions must be used
- GUID Partition Table (GPT) is a newer partition table (2010)
  - More space to store partitions
  - Used to overcome MBR limitations
  - 128 partitions max

Partitioning Tools

- **fdisk** has been around since the earliest Linux versions
  - Offers easy access to advanced features
  - Completely reworked to support MBR and GPT partitions
  - On new disks, create a GPT partition table using **g**
- **gdisk** was introduced to offer GPT support
  - Creates GPT partition table by default
- **parted** was introduced to make partitioning easier
  - Can be scripted easily
  - Advanced features are a bit hidden

Understanding Partition Type

- A partition type is a low-level identifier of the operating system, filesystem and boot manager using a specific partition
- Partition types were important in the past, currently partitions will work if the wrong partition type is set
  - In **fdisk**, use **t** to change the partition type
- Use Aliases to set the appropriate partition type:
  - **linux:** standard Linux
  - **swap:** linux swap
  - **uefi:** uefi boot
  - **lvm:** logical volume manager
- Other partition types are available, use **l** for a list

Understanding Linux Filesystems

- XFS is the default filesystem
    - Fast and scalable
    - Uses CoW to guarantee data integrity
    - Size can be increased, not decreased
- Ext4 was default in RHEL 6 and is still used
    - Backward compatible to Ext2
    - Uses Journal to guarantee data integrity
    - Size can be increased and decreased
- vfat offers multi-OS support
    - Used for shared devices
- Btrfs is a new and advanced filesystem
    - Not installed by default on RHEL

Creating Filesystems

- **mkfs.xfs** creates an XFS filesystem
- **mkfs.ext4** creates an Ext4 filesystem
- **mkfs.vfat** creates a vfat filesystem
- Use **mkfs.[Tab][Tab]** to show a list of available filesystems
- Do NOT use **mkfs** as it will create an Ext2 filesystem!

Mounting Filesystems

- After making the filesystem, you can mount it in runtime using the **mount** command
  - **mount /dev/vdb1 /mnt**
- Use **umount** before disconnecting a device
  - Use **lsof /mnt** if you get an error about open files
- The **mount** command shows all mounted filesystems, including administrative kernel mounts
- The **findmnt** command shows which filesystem is mounted where in the directory structure

Understanding /etc/fstab

- /etc/fstab is the main configuration file to persistently mount partitions
  - **/dev/sdb1 /data ext4 defaults 0 0**
- /etc/fstab content is used to generate systemd mounts by the **systemd-fstab-generator** utility
- To update systemd mounts, it is recommended to use **systemctl daemon-reload** after editing /etc/fstab

Avoiding Problems While Booting

- If an error was found in /etc/fstab, your system will drop a troubleshooting shell while booting
- After adding lines to /etc/fstab, use **mount -a** to mount all that hasn't been mounted yet
- Alternatively, use **findmnt --verify** to verify syntax

Labels and UUIDs

In datacenter environments, block device names may change. Different solutions exist for persistent naming

- **UUID:** a UUID is automatically generated for each device that contains a filesystem or anything similar
- **Label:** while creating the filesystem, the option **-L** can be used to set an arbitrary name that can be used for mounting the filesystem
- Unique device names are created in /dev/disk/*

Managing Persistent Naming Attributes

- **blkid** shows all devices with their naming attributes
- **tune2fs -L** is used to set a label on an Ext filesystem
- **xfs_admin -L** is used to set a label on an XFS filesystem
- **mkfs.* -L** is used to set a label while creating the filesystem

Managing **swap**

- Swap is RAM that is emulated on disk
- All Linux systems should have at least some swap
  - The amount of swap depends on the use of the server
- Swap can be created on any block device, including swap files
- Set partition type to linux-swap
- After creating the swap partition, use **mkswap** to create the swap FS
- Activate using **swapon**

Lab 4: Managing Partitions

To work on this lab, you'll need to create a 10GiB additional hard disk on your virtual machine. This disk needs to be completely available

- Create a primary partition with a size of 1GiB. Format it with Ext4, and mount it persistently on /mounts/files, using its UUID
- Create an extended partition that includes all remaining disk space. In this partition, create a 500MiB XFS partition and mount it persistently on /mounts/xfs, using the label myxfs
- Create a 500MiB swap partition and mount it persistently

You can evaluate your work using the **live-lab4-grade.sh** script in the course Git repository

Pearson

Day 2 Agenda

1. Users
2. Permissions
3. Processes
4. Networking
5. Systemd
6. Boot Procedure

Managing Users and Groups

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

The need for users

- A user is a security principle, user accounts are used to provide people or processes access to system resources
- Processes are using system accounts
- People are using regular user accounts

Creating and Managing Users

- **useradd**: create user accounts
- **usermod**: modify user accounts
- **userdel**: delete user accounts
- **passwd**: set passwords

Setting Defaults

- Default settings for creating users are in /etc/login.defs
- While creating a user, all files in /etc/skel will be copied to the user home directory

Limit Access

- User accounts can be temporarily locked
  - **usermod -L anna** will lock anna
  - **usermod -U anna** will unlock anna
- User accounts can be set to expire also
  - **usermod -e 2032-01-01 bill** expires user account bill on 01-01-2023
- Set /sbin/nologin as the shell for users that are not intended to log in at all
  - **usermod -s /sbin/nologin myapp**

Group Membership

- Each user must be a member of at least one group
- Primary Group Membership is managed through /etc/passwd
- The user primary group becomes group-owner if a user creates a file
- Additional (secondary) groups can be defined as well
- Secondary Group Membership is managed through /etc/groups
- Temporarily set primary group membership using **newgrp**
- Use **id** to see which groups a user is a member of

Create and Manage Groups

- Use **groupadd** to add groups
- **groupdel** and **groudmod** can be used to delete and modify groups
- Use **lid -g groupname** to list all users that are members of a specific group

Manage Password Settings

- Basic password requirements are set in /etc/login.defs
- To change password settings for current users, use **chage** or **passwd** as root

Lab 2: Manage Users and Groups

- Make sure that new users require a password with a maximal validity of 90 days
- Ensure that while creating users, an empty file with the name newfile is created to their home directory
- Create users anna, anouk, linda, and lisa
- Set the passwords for all users to 'password'
- Create the groups profs and students, and make users anna and anouk members of profs, and linda and lisa members of students

Use **live-lab2-grade.sh** to verify your solution

Managing Permissions

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

File Ownership

- To determine which permissions a user has, Linux uses ownership
- Every file has a user-owner, a group-owner and the others entity that is also granted permissions (ugo)
- Linux permissions are not additive, if you're the owner, permissions are applied and that's all
- Use **ls -l** to display current ownership and associated permissions
- Best practice: Set ownership before modifying permissions

Change File Ownership

- Use **chown user[:group] file** to set user-ownership
- Use **chgrp group file** to set group-ownership

Pearson

Permissions Overview

	On files	On directories
Read (4)	Open the file	List files and subdirectories
Write (2)	Modify the file	Create/Delete files and subdirectories
Execute (1)	Run the file	Use **cd** to activate the directory

Manage Permissions

- **chmod** is used to manage permissions
- It can be used in absolute or relative mode
- **chmod 750 myfile**
- **chmod +x myscript**

Special X

- When **x** is applied recursively, it would make directories as well as files executable
- In recursive command, use **X** instead
  - Directories will be granted the execute permission
  - Files will only get the execute permission if it is set already elsewhere on the file

Demo: Managing Permissions

- **mkdir -p /data/profs /data/students**
- **chown :profs /data/profs**
- **chgrp students /data/students**
- **chmod 770 /data/students**
- **chmod g+w,o-rx /data/profs**

Understanding Shared Group Directories

- If members of the same group need to share files within a directory, some special permissions are required
- The Set Group ID (SGID) permission ensures that all files created in the shared group directory are group owned by the group owner of the directory
- The sticky bit permission ensures that only the user who is owner of the file, or the directory that contains the file, is allowed to delete the file

Applying Shared Group Permissions

- **chmod g+s mydir** will apply SGID to the directory
- **chmod +t mydir** assigns sticky bit to the directory
- In absolute mode, a four digit number is used, of which the first digit is for the special permissions
- **chmod 3770 mydir** assigns SGID and sticky it, as well as rwx for user and group

Demo: Shared Group Permissions

- **su - anna**
- **touch /data/profs/anna{1..4}**
- **ls -l /data/profs/anna*; exit**
- **# chmod g+s /data/profs**
- **su - anna**
- **touch /data/profs/anna5; ls-l; exit**
- **su - anouk**
- **rm -f /data/profs/anna4; exit**
- **# chmod +t /data/profs**
- **su - anouk**
- **rm -f /data/profs/anna*; exit**

Understand **umask**

- The **umask** is a shell setting that subtracts the umask from the default permissions
  - Default is set in /etc/bashrc
  - Set user-specific overrides in ~/.bashrc
- Default permissions for file are 666
  - With umask 022, default permissions are 644
  - With umask 027 default permissions are 640
- Default permissions for directory are 777
  - With umask 022, default permissions are 755
  - With umask 027, default permissions are 750

Lab 3: Manage Permissions

- Create a shared group directory structure /data/profs and /data/students that meets the following conditions
  - Members of the groups have full read and write access to their directories, others has no permissions at all
- Modify default permission settings such that normal users have a umask that allows the user and group to read and write files and directories and others has read and execute on directories as well as read on new files

Managing Processes

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

Using Signals

- A signal allows the operating system to interrupt a process from software and ask it to do something
- Interrupts are comparable to signals, but are generated from hardware
- A limited number of signals can be used and is documented in **man 7 signal**
- Not all signals work in all cases
- The **kill** command is used to send signals to PID's
- You can also use **k** from top
- Different kill-like commands exist, like **pkill** and **killall**

Understanding Priority Management

- Linux Cgroups provide a framework to apply resource restrictions to Linux systems
- Cgroups can limit the amount of CPU cycles, available memory, and more
- If processes are equal from a perspective of Cgroups, the Linux **nice** and **renice** commands can be used to manage priority
- Cgroups configuration can change how **nice** behaves!

Managing **nice**

- If no specific Cgroups are defined, Linux **nice** and **renice** can be used to define CPU priority
- To change priorities of non-realtime processes, the **nice** and **renice** commands can be used
- Nice values range from -20 up to 19
- Negative nice value indicates an increased priority, a positive nice value indicates decreased priority
- Users can set their processes to a lower priority, to increase priorities you need root access
- **nice -n 19 dd if=/dev/zero of=/dev/null**
- Priority is always relative to other processes

Understanding System Tuning

- Kernel tunables are provided through the /proc/sys directory in the /proc pseudo file system
- Different files in the /proc/sys directory contain the current setting as its value
- Change the current value by echoing a new value into the file:
  - **cat /proc/sys/vm/swappiness**
  - **echo 40 > /proc/sys/vm/swappiness**
- To make settings persistent, write them to a file in /etc/sysctl.d:

```
cat >> swappiness.conf <<EOF
vm.swappiness = 40
EOF
```

Understanding **tuned**

- To make system tuning easier, **tuned** is provided
- **tuned** is a systemd service that works with different profiles
- **tuned-adm list** shows current profiles
- **tuned-adm profile virtual-guest** sets another profile as default
- Each profile contains a file with the name tuned.conf, that has a wide range of performance related settings
- The **reapply_sysctl = 1** parameter in /etc/tuned/tuned-main.conf ensures that, in case of conflict, the **sysctl** parameter wins

Demo: Using **tuned**

- **sudo dnf install -y tuned**
- **systemctl enable --now tuned**
- **tuned-adm list**
- **echo vm.swappiness = 33 > /etc/sysctl.d/swappiness.conf**
- **sysctl -p /etc/sysctl.d/swappiness.conf**
- **sysctl -a | grep swappiness**
- **mkdir /etc/tuned/myprofile**
- **cat >> /etc/tuned/myprofile/tuned.conf <<EOF**
  > [sysctl]
  > vm.swappiness = 66
  > EOF

Demo: Using **tuned**

- **tuned-adm profile myprofile**
- **tuned-adm profile**
- **sysctl -a | grep swappiness**
- **cat /etc/tuned/tuned-main.conf**

Pearson

Lab 6: Managing Processes

- Create a user linda and open a shell as this user
- As linda, run two background processes **sleep 600**; one of them with the highest possible priority, the other one with the lowest possible priority
- Use the most efficient way to terminate all current sessions for user linda

Managing Networking

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Device Names

- IP address configuration needs to be connected to a specific network device
- Use **ip link show** to see current devices, and **ip addr show** to check their configuration
- Every system has an **lo** device, which is for internal networking
- Apart from that, you'll see the name of the real network device, which is presented as a BIOS name

Host Name Resolution

- **hostnamectl set-hostname** is used to manage hostnames
- The hostname is written to /etc/hostname
- To resolve hostnames /etc/hosts is used
  - **10.0.0.11   server2.example.com server2**
- /etc/resolv.conf contains DNS client configuration
- The order of host name resolution is determined through /etc/nsswitch.conf

NetworkManager

- NetworkManager is the systemd service that manages network configuration
- Configuration is stored in files in /etc/NetworkManager/system-connections
  - Legacy files in /etc/sysconfig/network-scripts are still supported but deprecated
- Different applications are available to interface with NetworkManager
  - **nmcli** is a powerful command line utility
  - **nmtui** offers a convenient text user interface

Connections and Devices

- In NetworkManager, devices are network interfaces
- Connections are collections of configuration settings for a device, stored in the configuration file in /etc/NetworkManager/system-connections
- Only one connection can be active for a device

Managing the Network

- Use **nmtui** to manage network settings on the exam
- **nmcli** is definitely awesome, but much harder

Lab 7: Managing Network Configuration

- Set the hostname for your server to rhcsaserver.example.com
- Set your server to a fixed IP address that matches your current network configuration
- Also set a second IP address 10.0.0.10/24 on the same network interface
- Enable host name resolution for your local server hostname
- Reboot and verify your network is still working with the new settings

Systemd

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

Understanding Systemd Units

- Systemd is started as the first process after loading the kernel and is the manager of everything
- It is used for starting services like the secure shell server, a web server
- Apart from services, systemd takes care of many more items
- The items started by systemd are referred to as units
- **systemctl** is the main management tool for systemd
- To get an overview of all types of units that are available, use **systemctl -t help**

Understanding Unit Types

- Service units are used to start processes
- Socket units monitor activity on a port and start the corresponding Service unit when needed
- Timer units are used to start services periodically
- Path units can start Service units when activity is detected in the file system
- Mount units are used to mount file systems
- Other unit types are available, though less relevant for RHCSA

Understanding Service Units

- Service units are often used to start daemon processes
- Other types of Service units are available as well
  - **Type=simple** can be used to start any command through systemd

Pearson

Demo: Using **systemctl**

- **systemctl [Tab][Tab]**
- **systemctl -t help**
- **systemctl list-unit-files**
- **systemctl list-units [-t service]**

Systemd Unit Management Tasks

- **systemctl status** shows the current status of any unit
  - **systemctl status sshd**
- The **Active:** line in the output shows the current status
- The **Loaded:** line in the output shows which configuration is loaded, and whether the unit is enabled for automatic starting

Systemd Unit Management Tasks

- Use **systemctl status** to get current status
- **systemctl start** will start a unit that is not currently active
- **systemctl stop** will stop the unit
- **systemctl enable [--now]** is used to flag the unit for automatic starting upon system start
- **systemctl disable [--now]** is used to flag the unit to be no longer automically started
- **systemctl reload** will reload the unit configuration without restarting the unit
- **systemctl restart** restarts the unit after which the process it manages gets a new PID

Modifying Systemd Unit Configuration

- Default system-provided systemd unit files are in /usr/lib/systemd/system
- Custom unit files are in /etc/systemd/system
- Run-time automatically-generated unit files are in /run/systemd
- While modifying a unit file, do NOT edit the file in /usr/lib/systemd/system but create a custom file in /etc/systemd/system that is used as an overlay file
- Better: use **systemctl edit unit.service** to edit unit files
- Use **systemctl show** to show available parameters
- Using **systemctl reload** may be required

Demo: Modifying Units

- **dnf install httpd**
- **systemctl cat httpd.service**
- **systemctl show httpd.service**
- **systemctl edit httpd.service**
  - **Restart=always**
  - **RestartSec=5s**
- **systemctl daemon-reload**
- **systemctl restart httpd**
- **killall httpd**
- **systemctl status httpd**

Understanding **systemctl mask**

- Some units cannot work simultaneously on the same system
- To prevent administrators from accidentally starting these units, use **systemctl mask**
- **systemctl mask** links a unit to the /dev/null device, which ensures that it cannot be started
- **systemctl unmask** removes the unit mask

Understanding RHEL 9 Logging

- The systemd journal is the primary system for logging
- The journal forwards log messages to rsyslogd
- rsyslogd adds functionality
  - Persistency
  - Centralized log servers
  - Modules to send logs to specific destinations

Viewing Systemd Journal Messages

- **systemctl status *name.unit*** provides easy access to the last messages that have been logged for a specific service
- **journalctl** prints the entire journal
  - Important messages are shown in red
- **journalctl -p err** shows only messages with a priority error and higher
- **journalctl -f** shows the last 10 lines, and adds new messages while they are added
- **journalctl -u sshd.service** shows messages for sshd.service only

The Need for Persistency

- The systemd journal is non-persistent
- Persistency is taken care of by the rsyslog service
- Rsyslog offers all the filtering you need to fine-tune log persistency
- If desired, the systemd journal can be made persistent as well

Making the Journal Persistent

- Systemd journal settings are in /etc/systemd/journal.conf
- The setting **Storage=auto** ensures that persistent storage is happening automatically after creating the directory /var/log/journal
- Other options are:
  - **persistent**: stores journals in /var/log/journal
  - **volatile**: stores journals in the temporary /run/log/journal directory
  - **none**: doesn't use any storage for the journal at all

Demo: Making the Journal Persistent

- **grep 'Storage=' /etc/systemd/journal.conf**
- **mkdir /var/log/journal**
- **systemctl restart systemd-journal-flush**
- **ls /var/log/journal**

Lab 8: Working with systemd

- Make sure the httpd service is automatically started
- Edit its configuration such that on failure, it will be restarted after 1 minute

Managing LVM

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

RHEL Advanced Storage Solutions

- LVM Logical Volumes
  - Used during default installation of RHEL
  - Adds flexibility to storage (resize, snapshots, and more)
- Stratis
  - Next generation Volume Managing Filesystem that uses thin provisioning by default
  - Implemented in user space, which makes API access possible
- Virtual Data Optimizer
  - Focused on storing files in the most efficient way
  - Manages deduplicated and compressed storage pools
  - Now Integrated in LVM

LVM Elements

- **physical volume**: represents the storage device. Storage devices can be complete disks or partitions
- **volume group**: basic unit that represents all available storage. Foundation for creating logical volumes
- **logical volume**: the block device on which the file system will be created.

LVM Creating Procedure Overview

- Create a partition and set partition type to **lvm**
- Use **pvcreate /dev/sdb1** to create the physical volume
- Use **vgcreate vgdata /dev/sdb1** to create the volume group
- Use **lvcreate -n lvdata -L 1G vgdata** to create the logical volume
- Use **mkfs /dev/vgdata/lvdata** to create a filesystem
- Put in /etc/fstab to mount it persistently

Understanding Extents

- Extents are the elementary blocks of LVM allocation
- The extent size can be defined while defining the volume group
- Use **vgcreate -s 8M vgdata /dev/sdb1** to create a volume group with an extent size of 8MB
- All of the LVM logical volumes built from the volume group will use the same extent size
- Use **vgdisplay** to show properties of volume groups, including the extent size

Resizing Logical Volumes

- Use **vgs** to verify the volume group has unused disk space
- If required, use **vgextend** to add one or more PVs to the VG
- Use **lvextend -r -L +1G** to grow the LVM logical volume, including the filesystem it's hosting
  - **resize2fs** is a resize utility for Ext filesystems
  - **xfs_growfs** can be used to grow an XFS filesystem
- Shrinking is possible on Ext4 filesystems, not on XFS

Demo: Resizing a Logical Volume

- Create 2 partitions with a size of 1 GiB each and set the **lvm** partition type
- **vgcreate vgfiles /dev/sde1**
- **lvcreate -l 255 -n lvfiles /dev/vgfiles**
- **mkfs.ext4 /dev/vgfiles/lvfiles**
- **df -h**
- **vgs** shows no available extents in the volume group
- **vgextend vgfiles /dev/sde2** adds a new PV to the VG
- **lvextend -r -l +50%FREE /dev/vgfiles/lvfiles** is now possible
- **df -h** shows the newly available space in the volume

Lab 9: Managing Logical Volumes

To perform the tasks in this lab, you need 6 GiB of unpartitioned disk space to be available

- Create an LVM logical volume with the name lvdb and a size of 1020 MiB. Also create the VG and PV that are required for this LV
- Format this LV with the XFS filesystem and mount it persistently on /mounts/lvdb
- Grow your root logical volume with 5056 MiB

Boot Procedure

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Modifying Grub2 Runtime Parameters

- From the Grub2 boot menu, press e to edit runtime boot options to the end of the line that starts with **linux**
  - **systemd.unit=emergency.target**
  - **systemd.unit=rescue.target**
- Press c to enter the Grub2 command mode
  - From command mode, type **help** for an overview of available options

Modifying Grub2 Persistent Parameters

- To edit persistent Grub2 parameters, edit the configuration file in /etc/default/grub
- After writing changes, compile changes to grub.cfg
  - **grub2-mkconfig -o /boot/grub2/grub.cfg** on BIOS
  - **grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg** on UEFI

Understanding Systemd Targets

- A systemd target is a group of unit files
- Some targets are *isolatable*, which means that they define the final state a system is starting in
  - emergency.target
  - rescue.target
  - multi-user.target
  - graphical.target
- When enabling a unit, it is added to a specific target

Setting the Default Systemd Target

- Use **systemctl get-default** to see the current default target
- Use **systemctl set-default** to set a new default target

Booting into a Specific Target

- On the Grub 2 boot prompt, use **systemd.unit=xxx.target** to boot into a specific target
- To change between targets on a running system, use **systemctl isolate xxx.target**

Lab 10: Managing the Boot Procedure

- Configure your system to boot in a multi-user target by default
- Persistently remove the options that hide startup messages while booting

Day 3 Agenda

1. Remote Mounts and Autofs
2. SELinux
3. Firewalling
4. Time
5. Scheduling Tasks

Remote Mounts and Autofs

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

Demo: Configuring a Base NFS Server

- **dnf install nfs-utils**
- **mkdir -p /nfsdata /home/ldap/ldapuser{1..9}**
- **echo "/nfsdata *(rw,no_root_squash)" >> /etc/exports**
- **echo "/home/ldap *(rw,no_root_squash) >> /etc/exports**
- **systemctl enable --now nfs-server**
- **for i in nfs mountd rpc-bind; do firewall-cmd --add-service $i --permanent; done**
- **firewall-cmd --reload**

Mounting NFS Shares

- Make sure that **nfs-utils** is installed
- Use **showmount -e nfsserver** to show exports
- Use **mount nfsserver:/share /mnt** to mount

Understanding Automount

- In **/etc/auto.master** you'll identify the directory that automount should manage, and the file that is used for additional mount information
  - **/data /etc/auto.nfsdata**
- In **/etc/auto.nfsdata** you'll identify the subdirectory on which to mount, and what to mount exactly
  - **files -rw nfsserver:/nfsdata**
- Ensure the autofs service is started:
  - **systemctl enable --now autofs**

Tip: check /etc/auto.misc for syntax examples on the exam

Understanding Wildcard Mounts

- Automount is common for home directory access
- In this scenario, an NFS server is providing access to homedirectories, and the homedirectory is automounted by a user while logging in
- To support different directory names in one automount line, wildcards are used
- ***   -rw nfsserver:/home/ldap/&**

Lab 11: Setting up Automount

- To perform this lab it is recommended to use a second server with the name **nfsserver**. If that's not possible, configure the NFS server parts on localhost
- On **nfsserver**, create home directories /home/ldap/ldapuser1 .. ldapuser9
- Configure **nfsserver** to NFS export these home directories
- Automount the home directories on /homes/. The result should be that /homes/ldapuser*n* is accessed, the corresponding directory from **nfsserver** is mounted

SELinux

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Understanding the Need for SELinux

- Linux security is built on UNIX security
- UNIX security consists of different solutions that were never developed with current IT security needs in mind
- Most of these solutions focus on a part of the operating system
- SELinux provides a complete and mandatory security solution
- The principle is that if it isn't specifically allowed, it will be denied
- As a result, "unknown" services will always need additional configuration to enable them in an environment where SELinux is enabled
- Known services work well with the standard SELinux configuration

Managing SELinux States

- **getenforce** will show the current state
- **setenforce** toggles between Enforcing and Permissive
- Use **selinux=0** as a Grub kernel boot argument to set SELinux to disabled

Managing SELinux States while Booting

- SELinux state can be set at boot time using a kernel parameter
  - **enforcing=0** will start in permissive mode
  - **enforcing=1** will start in enforcing mode
  - **selinux=0** disabled SELinux
  - **selinux=1** enabled SELinux
- Access the Grub bootloader prompt to change the settings while booting

Exploring SELinux Components

- SELinux works with context labels
- Context labels define specific permissions
- Context labels are applied to source objects and target objects
- Source objects are
  - Users
  - Processes
- Target objects are
  - Files and directories
  - Ports
- The SELinux policy has many rules to define which source context has access to which target context

Managing SELinux

- Context management is about applying contexts to mostly files, directories and ports
- You'll need to apply a context that matches a specific rule
- Booleans allow parts of the SELinux policy to be rewritten to allow or disallow specific functionality in an easy way

Understanding Context Labels

- Every object is labeled with a context label
  - **user:** user specific context
  - **role:** role specific context
  - **type:** flags which type of operation is allowed on this object
- In most configurations, only context type matters, so you can safely ignore user and role for RHCSA
- Many commands support a **-Z** option to show current context information
- Context types are used in the rules in the policy to define which source object has access to which target object

Understanding Default File Context

- Most service don't need additional SELinux configuration if default settings are used
- When files are created in a directory, they typically inherit the context of the parent directory
- When files are copied, they typically inherit the context of the parent directory
- When files are moved, they keep the original context

Managing File Context Labels

- Use **semanage fcontext** to set the file context label
  - This will write the context to the SELinux Policy but it is not written yet to the filesystem
  - Use **semanage fcontext -a** to set a new context label
  - Use **semanage fcontext -m** to modify an existing context label
- To enforce the policy setting on the file system, use **restorecon**
- Alternatively, use **touch /.autorelabel** to relabel all files to the context that is specified in the policy
- See **man semanage-fcontext** for documentation
- Do NOT use **chcon** as the changes it makes may be overwritten
- Use **semanage fcontext -l -C** to show only settings that have changed in the current policy

Finding the Right Context

- If you apply a non-default configuration, check the default configuration context setting
- Read man pages from **selinux-policy-doc**
- Use **sealert** (covered later)

Managing Ports

- Network ports are also provided with an SELinux context label
- The SELinux policy is configured to allow default port access
- For any non-default port access, use **semanage port** to apply the right label to the port
- Use the examples section in **man semanage-port** for examples

Using Booleans

- A boolean is an easy-to-use configuration switch to enable or disable specific parts of the SELinux policy
- For an overview of all booleans, use **semanage boolean -l** or **getsebool -a**
- To set booleans, use **setsebool -P *boolean* [on|off]**
- Use **semanage boolean -l -C** to see all booleans that have a non-default setting

Using **sealert**

- SELinux uses **auditd** to write log messages to the audit log
- Messages in the audit log may be hard to interpret
- Ensure that **sealert** is available, it interprets messages from the audit log, applies SELinux AI, and writes meaningful messages to /var/log/messages
- Run the **sealert** command, including the UUID message to get advice on how to troubleshoot specific issues

Troubleshooting SELinux

- If you think that SELinux is blocking access, start by using **setenforce 0** and try again. If it works now, you have confirmed that SELinux is blocking the requested activity
- Use **grep AVC /var/log/audit/audit.log** to see raw audit messages. Look at the source context and target context
- Install selinux-policy-doc, using **dnf install selinux-policy-doc** for additional man pages, and try to understand what you need to do
- Confirm by using **journalctl | grep sealert** and read the alert message that was generated

Lab 12: Managing SELinux

- Configure the Apache web server to bind to port 82
- Use **mv /etc/hosts /var/www/html/** and ensure that the file gets an SELinux context that makes it readable by the Apache web server

Firewalling

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Understanding RHEL Firewalling

- The Linux kernel provides the netfilter framework to take care of firewall related network operations:
  - packet filtering
  - network address translation
  - port forwarding
- Netfilter forwards specific operations to kernel modules
- **nftables** is the framework that applies firewalling
- **firewalld** is a service, managed by systemd, which RHEL uses as the front end to manage **nftables** firewalls

Understanding Firewalld Components

Firewalld is using different components to make firewalling easier

- **Service:** the main component, contains one or more ports as well as optional kernel modules that should be loaded
- **Zone:** a default configuration to which network cards can be assigned to apply specific settings
- **Ports:** optional elements to allow access to specific ports
- Additional components are available as well, but not frequently used in a base firewall configuration

Using **firewall-cmd**

- The **firewall-cmd** command is used to write firewall configuration
- Use the option **--permanent** to write to persistent (but not to runtime)
- Without **--permanent** the rule is written to runtime (but not to persistent)

Demo: Allowing Incoming HTTP traffic

- **systemctl status firewalld**
- **firewall-cmd --list-all**
- **firewall-cmd --get-services**
- **firewall-cmd --add-service http**
- **firewall-cmd --add-service http --permanent**

Lab 13: Configuring a Firewall

- Configure **firewalld** such that remote access to the SSH and FTP processes is allowed. Make sure the configuration is applied immediately as well as persistently

Time

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Exploring Linux Time

- While booting, the system gets its time from the hardware clock
- System time is set next, according to the hardware clock
- Internet time can be used to synchronize time

Managing Linux Time

- **hwclock** is used to set hardware time
- Also use it to synchronize time
  - **hwclock --systohc**
  - **hwclock --hctosys**
- **date** is used to show and set time
- **timedatectl** is used to manage time and time zone configuration

Using **timedatectl**

- **timedatectl** is a new utility that allows you to manage all aspects of system time
  - **timedatectl status** will show all time properties currently used
  - **timedatectl set-time** is used to change the time
  - **timedatectl set-timezone** is used to change the timezone
  - **timedatectl set-ntp** enables or disables NTP time synchronization
- To synchronize time using NTP, an NTP service must be configured
  - RHEL 9 uses **chronyd**
  - **systemd-timesyncd.service** is another NTP service - not currently used on RHEL

Pearson

Managing an NTP Client

- **chronyd** is the default RHEL 9 NTP service
- Do NOT use **timedatectl set-ntp-servers**
- Use /etc/chrony.conf to specify synchronization parameters
  - **pool 2.rhel.pool.ntp.org iburst** configures a pool of NTP servers
  - **server myserver.example.com** configures a single NTP time source
  - Use **iburst** to permit fast synchronization
- After modifying its contents, use **systemctl restart chronyd** to restart the **chronyd** service
- Use **chronyc sources** to verify proper synchronization

Lab 14: Configuring Time Services

- Configure your system to synchronize time with the servers in pool.ntp.org

Scheduling Tasks

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Pearson

RHEL 9 Scheduling Options

- **systemd** timers are the primary solution for scheduling recurring jobs on RHEL 9
- **crond** is an older scheduling solution which is still supported and a bit easier to schedule custom tasks
- **at** is available to schedule non-recurring user tasks

Understanding **systemd** Timers

- Systemd provides unit.timer files that go together with unit.service files to schedule the service file
- When using **systemd** timers, the timer should be enabled / started, and NOT the service unit
- **systemd** timers are often installed from RPM packages
- In the timer unit file, the **OnCalendar** option specifies when the service should be started
- On RHEL 9, **systemd** timers are the default way for scheduling recurring services

Demo: Analyzing **systemd** Timers

- **systemctl list-units -t timer**
- **systemctl list-unit-files logrotate.***
- **systemctl status logrotate.service**
- **systemctl status logrotate.timer**
- **dnf install -y sysstat**
- **systemctl list-unit-files sysstat.***
- **systemctl cat sysstat-collect.timer**

Understanding Timer Activation

- The **systemd** timer **OnCalendar** option uses a rich language to express when the timer should activate
  - **OnCalendar=*:00/10** runs every 10 minutes
  - **OnCalendar=2023-*-* 9:9,19,29:30** runs the service every day in 2023 at 9:09:30, 9:19:30 and 9:29:30
- Use **OnUnitActivateSec** to start the unit a specific time after the unit was last activated
- Use **OnBootSec** or **OnStartupSec** to start the unit a specific time after booting
- Read **man 7 systemd.time** for specification of the time format to be used, use **man 7 systemd.timer** for generic information about timers

Demo: Managing **systemd** Timers

- **systemctl daemon-reload**
- **systemctl start touchfile.timer**
- **systemctl status touchfile.service**
- **watch ls -l /tmp/myfile.txt**
- **systemctl stop touchfile.timer**

Understanding **cron**

- **cron** is an old UNIX scheduling option
- It uses **crond**, a daemon that checks its configuration to run cron jobs periodically
- Still on RHEL 9, **crond** is enabled as a **systemd** service by default
- Most services that need scheduling are scheduled through **systemd** timers

Using **cron**

- The **cron** service checks its configuration every minute
- **/etc/crontab** is the main (managed) configuration file
- **/etc/cron.d/** is used for drop-in files
- **/etc/cron.{hourly,daily,weekly,monthly}** is used as a drop-in for scripts that need to be scheduled on a regular basis
  - Make sure these scripts have the execute bit set!
- User specific cron jobs can be created using **crontab -e**

Understanding Cron Time Specification

- Cron time specifications are specified as minute, hour, day of month, month, day of week
- **0 * * dec 1-5** will run a cron job every monday thru friday on minute zero in December
- The /etc/crontab file has a nice syntax example
- Do NOT edit /etc/crontab, put drop-in files in /etc/cron.d instead

Day 4 Agenda

1. Troubleshooting
2. Containers
3. Optional Exam Practice Lab

Troubleshooting

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Changing a Lost Root User Password

- Enter Grub menu while booting
- Find the line that loads the Linux kernel and add **init=/bin/bash** to the end of the line
- **mount -o remount,rw /**
- **passwd root**
- **touch /.autorelabel**
- **exec /usr/lib/systemd/systemd**

Containers

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Understanding Containers

- A container is like an app on a smartphone; it's a package that contains all that is needed to run an application
- This makes containers the solution for the dependency challenge
- Containers are started from container images
- Images are provided through image registries

Containers and Linux

- Containers rely on features provided by the Linux operating system
  - Control groups set limits to the amount of resources that can be used
  - Namespaces provide isolation to ensure the container only has access to its own data and configuration
  - SELinux enforces security

Understanding Rootless Containers

- Containers need a user ID to be started on the host computer
- Root containers are started by the root user: they should be avoided
- Rootless containers are started as a non-root user
  - Rootless containers can generate a UID dynamically, or be preconfigured to use a specific UID
- Rootless containers have a few considerations
  - No unlimited access to the filesystem
  - Can't bind to privileged network ports

Using Images and Registries

- Container images are used to package container applications with all of their dependencies
- Images are built according to the Open Containers Initiative (OCI) specification
- The OCI standard guarantees compatibility so that images can be used in different environments, like **podman** on RHEL or **docker**
- Container images are offered through registries

Using Registries

- Public registries such as hub.docker.com provide access to community-provided container images
- Private registries can be created to host container images internally
- Community images optimized for use in Red Hat environments are provided through quay.io
- Red Hat distributes certified images that are accessible only with Red Hat credentials
  - registry.redhat.io is for official Red Hat products
  - registry.access.redhat.com is for third-party products
- Red Hat container catalog (https://catalog.redhat.com) is a web interface to the Red Hat images

Accessing Red Hat Registries

- Red Hat registries can be accessed with a Red Hat account
- Developer accounts (https://developers.redhat.com) do qualify
- Use **podman login registry.redhat.io** to login to a registry
- Use **podman login registry.redhat.io --get-login** to get your current login credentials

Configuring Registry Access

- Registry access is configured in /etc/containers/registries.conf
- A user specific registries.conf file can be created as ~/.config/containers/registries.conf
- Red Hat recommends you use Fully Qualified Container Names: don't use **podman run nginx**, but use **podman run docker.io/library/nginx** to avoid confusion

Using Containerfile

- A Containerfile (previously known as Dockerfile) is a text file with instructions to build a container image
- Containerfiles have instructions to build a custom container based on a base image such as the UBI image
- UBI is the Universal Base Image, an image that Red Hat uses for all of its products
- UBI is not the most efficient image, better use alpine if you want it to be tiny

Demo: Building an image from a Containerfile

- All demo's in this part are as non-root user
- **sudo dnf install container-tools**
- **git clone https://github.com/sandervanvugt/rhcsa9**
- **podman info**
- **cd rhcsa9**
- **podman images**
- **podman login registry.access.redhat.com**
- **podman build -t mymap .**
- **podman images**

Running Containers

- Use **podman run** to run a container image
  - It will search for the image in the configured registries
  - If found, it will pull the image and run the container
- Use **podman ps** to verify that the image currently is running
- If not seen, use **podman ps -a** to also show containers that have stopped
- Use **podman inspect** to see what is inside an image or a container

Understanding Container Commands

- When started with **podman run**, the container runs its default command
- To run an alternative command, it can often (not always) be specified as a command line argument
  - **podman run ubi8 sleep infinity**
- To run an image from a specific registry, specify the complete image name
- Command line options for the specific **podman** command need to be specified before the name of the image

Demo: Running Containers

- **podman search ubi**
- **podman run --name sleepy docker.io/redhat/ubi9 sleep 3600**
- from another terminal: **podman ps**
- **podman stop sleepy**
- **podman images**
- **podman run -d --name sleepy docker.io/redhat/ubi9 sleep 3600**
- **podman rm sleepy**
- **podman ps -a**

Troubleshooting Containers

- Troubleshooting containers is often troubleshooting the container primary command
- Notice that some containers run to completion and there's nothing really to troubleshoot if you don't see them!
- Use **podman inspect container** to see which command is started
- Use **podman run -it** ... to start the container with an interactive terminal
- Use **podman logs** to explore logs created by the container

Managing Environment Variables

- Some images require site-specific information to be passed while running
- Use **-e KEY=VALUE** to pass these values through environment variables while running the container

Demo: Using Environment Variables

- podman run --name mydb registry.access.redhat.com/rhel9/mariadb-1011
- podman ps -a
- podman logs mydb
- skopeo inspect docker://registry.access.redhat.com/rhel9/mariadb-1011
- podman rm mydb
- podman run --name mydb -e MYSQL_ROOT_PASSWORD=password registry.access.redhat.com/rhel9/mariadb-1011

Configuring Application Access

- Container access happens through port mappings
- A port on the container host is exposed and forwards traffic to the container port
- Port mappings can be set while starting the container, but not on a container that has already been started
- Rootless containers can only map to a non-privileged port (higher than 1024)

Demo: Mapping Ports

- Run as non-root user
- **podman run -d -p 80:80 docker.io/library/nginx**
- **podman run -d -p 8080:80 docker.io/library/nginx**
- **podman port -a**
- **sudo firewall-cmd --add-port=8080/tcp --permanent**
- **sudo firewall-cmd --reload**
- **sudo semanage port -l | grep 8080** explains why no additional SELinux action is needed

Managing Storage

- Container storage is ephemeral by nature
- Persistent storage can be provided by creating a directory on the container host and bind-mounting that directory in the container using **podman run -d … -v /hostdir:/containerdir**
- When bind-mounting directories on the host, the user used inside the container namespace needs to own the directory on the host
- Use **podman inspect** on the image to find out which user this is
- And use **podman unshare** to make that user who only exists in the namespace owner on the host as well (as a mapped user ID)

Why do we need **podman unshare**?

- Rootless containers are launched in a namespace
- The namespace provides isolation, allowing the container inside the namespace to have root access which does not exist outside the namespace
- UIDs used inside a container namespace are mapped to UIDs on the host which are dynamically created
  - The container UID exists on the host as $((CONTAINER_UID + 99999))
- The **podman unshare** command can be used to run commands inside the container namespace

Understanding Non-root User Mappings

- To set appropriate directory ownership on bind-mounted directories for rootless containers, additional work is required
- First, find the UID of the user that runs the main application: **podman inspect imagename**
- Use **podman unshare chown nn directoryname** to set the container UID as the owner of the directory on the host
  - Notice that **directoryname** must be in the user home directory because otherwise it wouldn't be part of the user namespace
- Verify the mapped user is owner on the host, using **ls -ld ~/directoryname**
- Verify within the namespace as well, using **podman unshare ls -ld ~/directoryname**

Demo: Bind Mounting in Rootless Containers

- Run as non-root user
- **podman search mariadb | grep ubi9**
- **mkdir ~/mydb** # must be in current host user homedir!
- **podman run -d --name mydb -e MYSQL_ROOT_PASSWORD=password -v /home/student/mydb:/var/lib/mysql registry.access.redhat.com/rhel9/mariadb-1011**
- **podman ps -a** # will show it has failed
- **podman inspect mydb** # find the user ID
- **podman unshare chown 27 mydb**
- **ls -ld mydb; podman unshare ls -ld mydb**

Pearson

Configuring SELinux for Shared Directories

- At this point ownership is set correctly, you'll next have to take care of SELinux before using **podman run -v …** to bind mount the directory
- To bind mount a host directory in the container, the **container_file_t** SELinux context type must be used
- If ownership on the host directory has been configured all right, use the **:Z** option to automatically set this context:
  - **podman run … -v /home/student/mydb:/var/lib/mysql:Z …**

Demo: Bind Mounting in Rootless Containers

- As file ownership has been taken care of in the preceding steps, you're now ready to bind mount, taking care of SELinux as well
- Notice that the directory *inside* the container is mapped to the homedir of the user that runs the container main application
- **podman stop mydb**
- **podman rm mydb**
- **podman run -d --name mydb -e MYSQL_ROOT_PASSWORD=password -v /home/student/mydb:/var/lib/mysql:Z registry.access.redhat.com/rhel9/mariadb-1011**
- **ls -Z mydb**

Summarizing Bind Mounts

- While using a bind-mount, the rootless container user needs access to the mapped directory on the host
- First, find the User defined in the container image, using **podman image inspect ...** Let's say it's UID 27
- Next, create the directory in the rootless user homedirectory: **mkdir /home/student/mydb**
- Then use **podman unshare chown 27 /home/student/mydb**
- Verify: **podman unshare ls -ld /home/student/mydb** and **ls -ld /home/student:mydb**
- Run the bind-mount, with the :Z option for SELinux: **podman run -d --name mydb -e MYSQL_ROOT_PASSWORD=password -v /home/student/mydb:/var/lib/mysql:Z registry.access.redhat.com/rhel9/mariadb-1011**

Autostarting Containers

- To automatically start containers in a stand-alone situation, you can create systemd user unit files for rootless containers and manage them with **systemctl**
- Alternatively, use the Docker compatible **--restart=always** option, which requires the **podman-restart** service to be enabled
- If Kubernetes or OpenShift is used, containers will be automatically started by default

Running Systemd Services as a User

- Systemd user services start when a user session is opened, and close when the user session is stopped
- Use **loginctl enable-linger** to change that behavior and start user services for a specific user (requires root privileges)
  - **loginctl enable-linger linda**
  - **loginctl show-user linda**
  - **loginctl disable-linger linda**

Managing Containers Using Systemd Services

- Create a regular user account to manage all containers
- Use **podman** to generate a user systemd file for an existing container
- Before, **mkdir ~/.config/systemd/user; cd ~/.config/systemd/user**
- Notice the file will be generated in the current directory
  - **podman generate systemd --name myweb --files --new**
- To generate a service file for a root container, do it from /etc/systemd/system as the current directory
- Tip: **man podman-generate-systemd**

Understanding **podman generate --new**

- The **podman generate --new** option will create a new container when the systemd unit is started, and delete that container when the unit is stopped
- Without the **--new** option, the container is not newly created or deleted when it is stopped

Creating User Unit Files

- Use **podman generate** to create user-specific unit files in ~/.config/systemd/user
- Manage them using **systemctl --user**
  - **systemctl --user daemon-reload**
  - **systemctl --user enable myapp.service** (requires linger)
  - **systemctl --user start myapp.service**
- **systemctl --user** commands only work when logging in on console or SSH and do not work in sudo and su sessions

Demo: Autostarting User Containers

- **sudo useradd linda; sudo passwd linda**
- **sudo loginctl enable-linger linda**
- **ssh linda@localhost**
- **mkdir -p ~/.config/systemd/user; cd ˜/.config/systemd/user**
- **podman run -d --name mynginx -p 8081:80 nginx**
- **podman ps**
- **podman generate systemd --name mynginx --files --new**
- **systemctl --user daemon-reload**
- **systemctl --user enable container-mynginx.service**
- **sudo reboot**
- After reboot: **ps faux | less** # search for processes owned by linda

Surviving all Challenges

- Log in as the user that should start the container, do NOT use **su -**
  - set a password to do so
- As that user, **mkdir -p ~/.config/systemd/user; cd ~/.config/systemd/user**
- From that directory: **podman generate systemd --new --files**
- Don't forget **man podman-generate-systemd**

Lab 15: Managing Containers

- Ensure that you have full access to the Red Hat container repositories
- Run a Mariadb container, based on the registry.redhat.io/rhel9/mariadb-105 image, which meets the following conditions
  - The container is started as a rootless container by user student
  - The container must be accessible at host port 3206
  - The database root password should be set to password
  - The container uses the name mydb
  - A bind-mounted directory is accessible: the directory /home/student/mariadb on the host must be mapped to /var/lib/mysql in the container
- The container must be configured such that it automatically starts as a user systemd unit upon start of the computer

To evaluate your work, use **live-lab15-grade.sh**

Sample Exam

Sander van Vugt
Author, Public Speaker, Trainer,
Certified Technical Consultant

Exploring RHCSA Practice Exam Assignments

- **Check Git Repo at https://github.com/sandervanvugt/rhcsa9**

Additional Learning

- Check the resources.txt file in the course Git repository: https://github.com/sandervanvugt/rhcsa9